Sissejuhatus infotehnoloogiasse

Järgmiste loengute struktuur

Loengud algavad riistvara tasemelt (alates transistorist) ja liiguvad üha suurema abstraktsuse suunas:

- Riistvara
- Protsessori programmeerimine, assembler
- Kõrgkeeled: mugavam programmeerimine
- Suurte rakendussüsteemide kokkupanek, opsüsteem, komponentide kasutamine
- Võrgurakenduste kokkupanek: hulga arvutite kui rakenduse komponentide kasutamine
- Eriti võimsad kõrgkeeled, funktsionaalne ja loogiline programmeerimine
- Teooriat: keerukus, lahenduvus: mida saab kui ruttu arvutada, mida saab üldse arvutada
- Tehisintellekt
- IT äri ja juhtimine: kuidas raha saada

Olulisi põhimõtteid abstraktsioonide osas

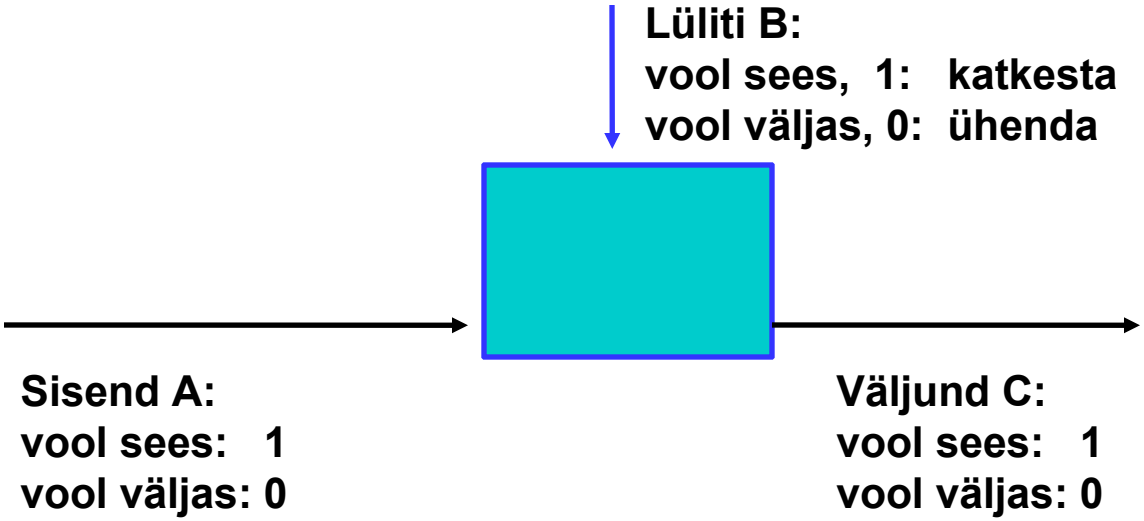
- Kõrgkeeled, komponendid, võrguvärk jms on vajalik ainult selleks, et arendaja saaks rakendusi kiiremini teha.
- Arendaja mõtleb abstraktsioonide tasemel, aga lõpuks töötab kõik ikkagi transistoridel.
- **Abstraktsioonid nö “tilguvad läbi”**: iga abstraktsiooni juures on vaja põhimõtteliselt aru saada, kuidas töötab alumine, vähem abstraktne tase. Alati on vahel vaja midagi allpool teha!!!
- Arendajal on reeglina kalduvus üle-abstraheerida! Süsteemi ehitamisel tuleb ise, meelega vähem abstraheerida, kui tahaks. Näiteks ei õnnestu enamasti tarkvara taaskasutus jne jne ...
- Loe juurde: The law of leaky abstractions: http://www.joelonsoftware.com/articles/LeakyAbstractions.html

Ülevaade loengust

- **Riistvara**:
  - Riistvara komponendid
  - Protsessori tööpõhimõte
- Programmeerimiskeelte hierarhia ja mehaanika (jätkub järgmine loeng)
  - **Assembler**
  - C
    - C ja assembler
    - C ja mäluhaldus
  - Kõrgema taseme keeled
    - Prahikoristus
    - Listid
    - ....

Komponendid

- Peamine idee: **transistorid kui “katkestusmootoriga” lülitid**

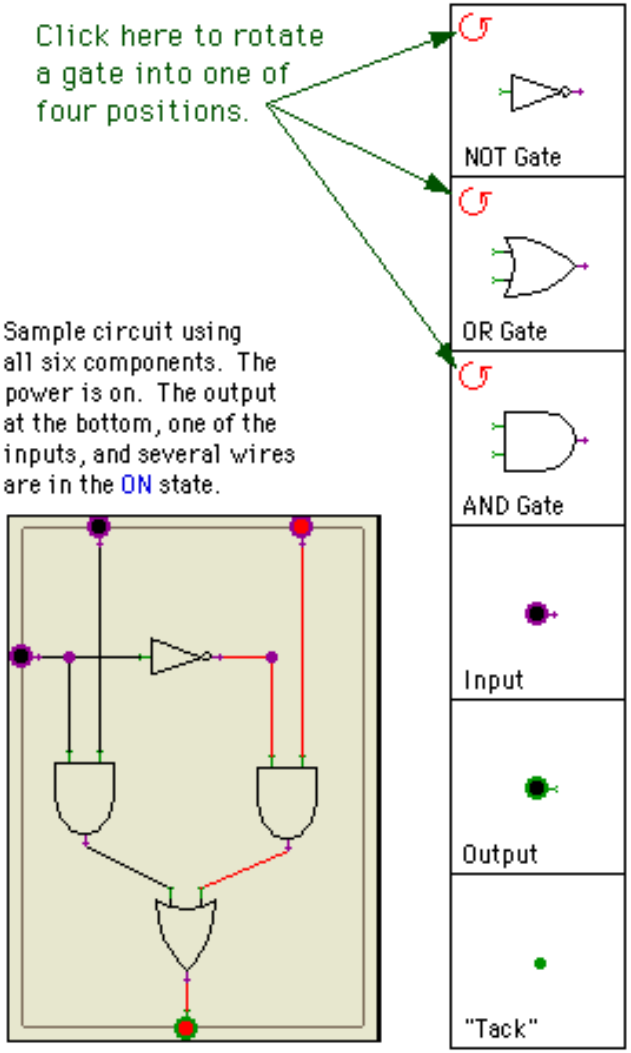


C = (A and -B)

- Väikestest komponentidest ehitatakse suuremaid, millest omakorda veel suuremaid.
- Komponendid on kui mustad kastid: teame nende väljundit vastava sisendi korral, aga enamasti mitte nende tehnilist sisu.

Komponendid (Eck)

(A and C) or (B and (not C))



NOT gate turns its output ON when its input is OFF, and vice versa. A NOT gate reverses its input.

AND gate turns its output ON when both of its inputs are ON. Otherwise, the output is OFF.

OR gate turns its output ON when either of its inputs is on (or when both are ON).

Inputs can be placed around the edges of a circuit to provide input values for the circuit as a whole.

Outputs can be placed around the edges of a circuit to represent values computed by the circuit as a whole.

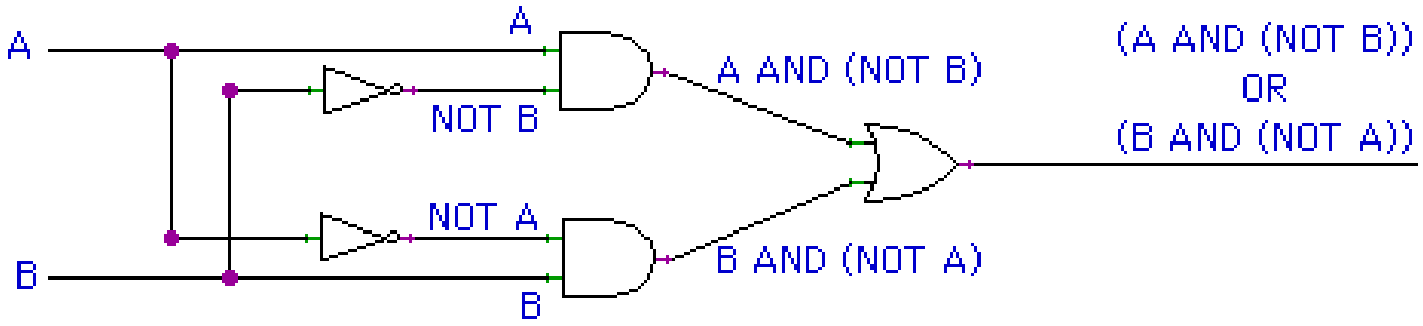
Tacks are simply connecting points for wires, which can be used to help make your circuits neater.

Komponendid (Eck)

NB! Loe ja harjuta nende simulaatoritega:

- http://math.hws.edu/TMCM/java/labs/xLogicCircuitsLab1.htm
- http://math.hws.edu/TMCM/java/labs/xLogicCircuitsLab2.html

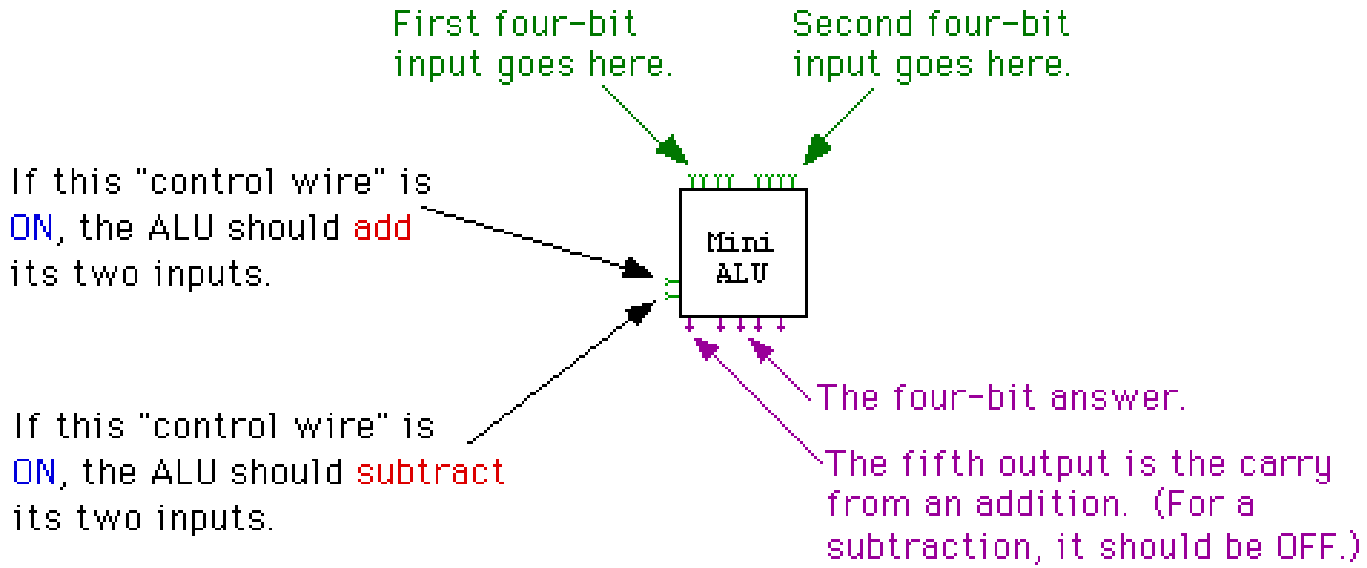
(A and (not B)) or (B and (not A))



Neljabitine liitja (four-bit adder)

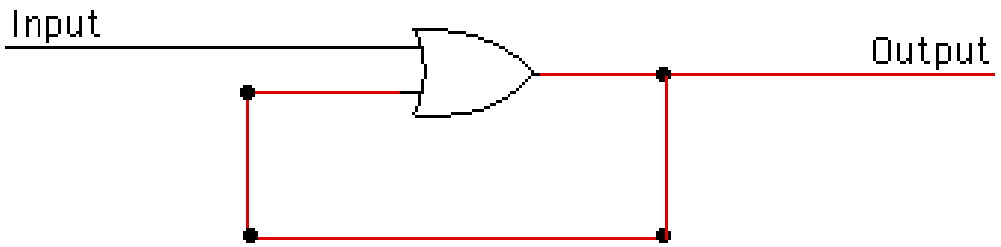
- Kaheksa pluss kaks sisendjuhet, neli pluss üks väljundjuhet

1011	1111	1111	1010	0111	0001
0110	0001	1111	0101	1010	0011
-----	-----	-----	-----	-----	-----
10001	10000	11110	01111	10001	00100

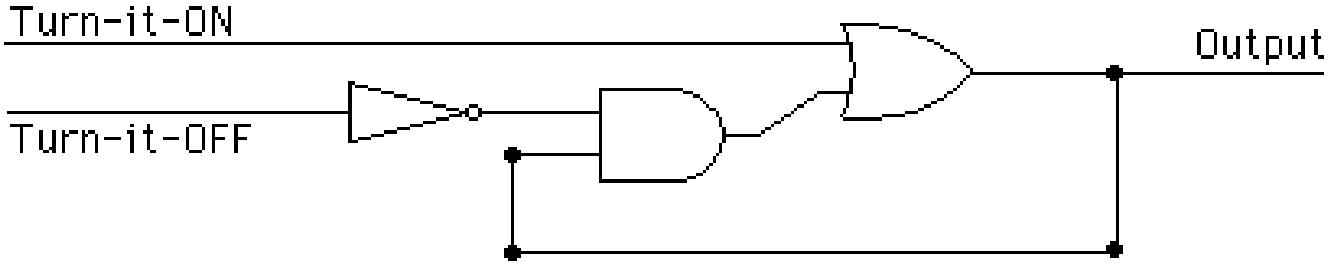


Mälu

- Tagasiside

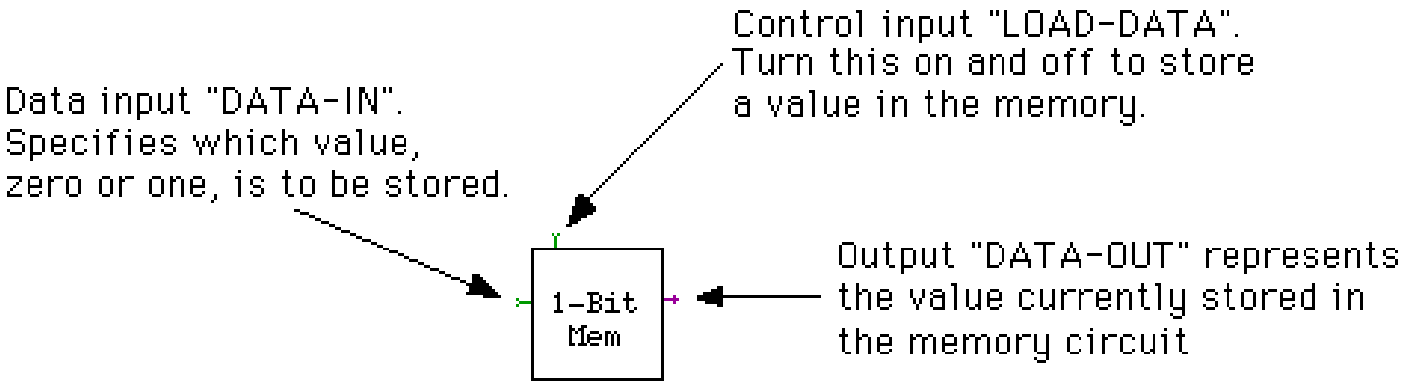


- Lülitatav tagasiside: triger



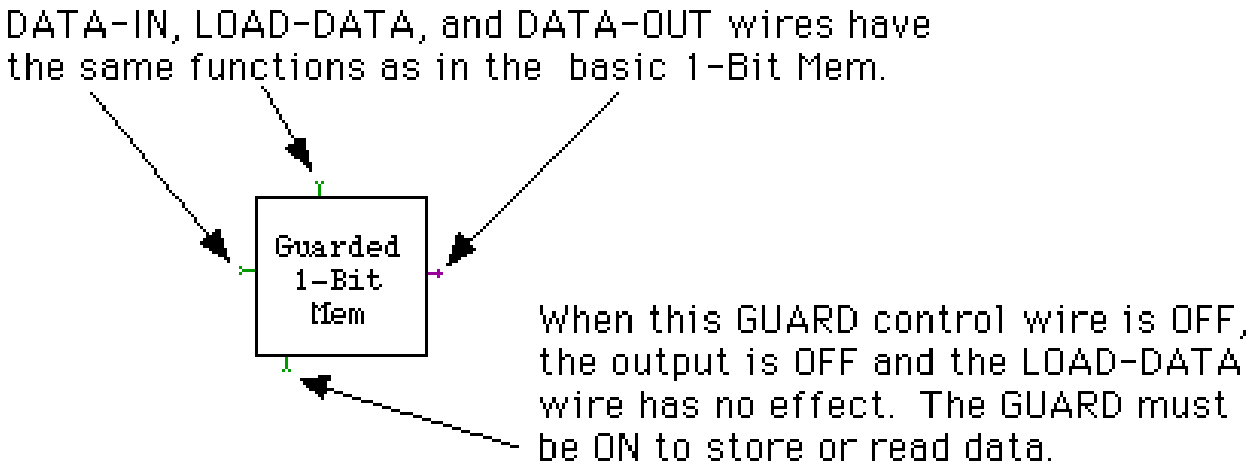
Ühebitine mälukiip

- Kaks sisend- ja üks väljundjuhe



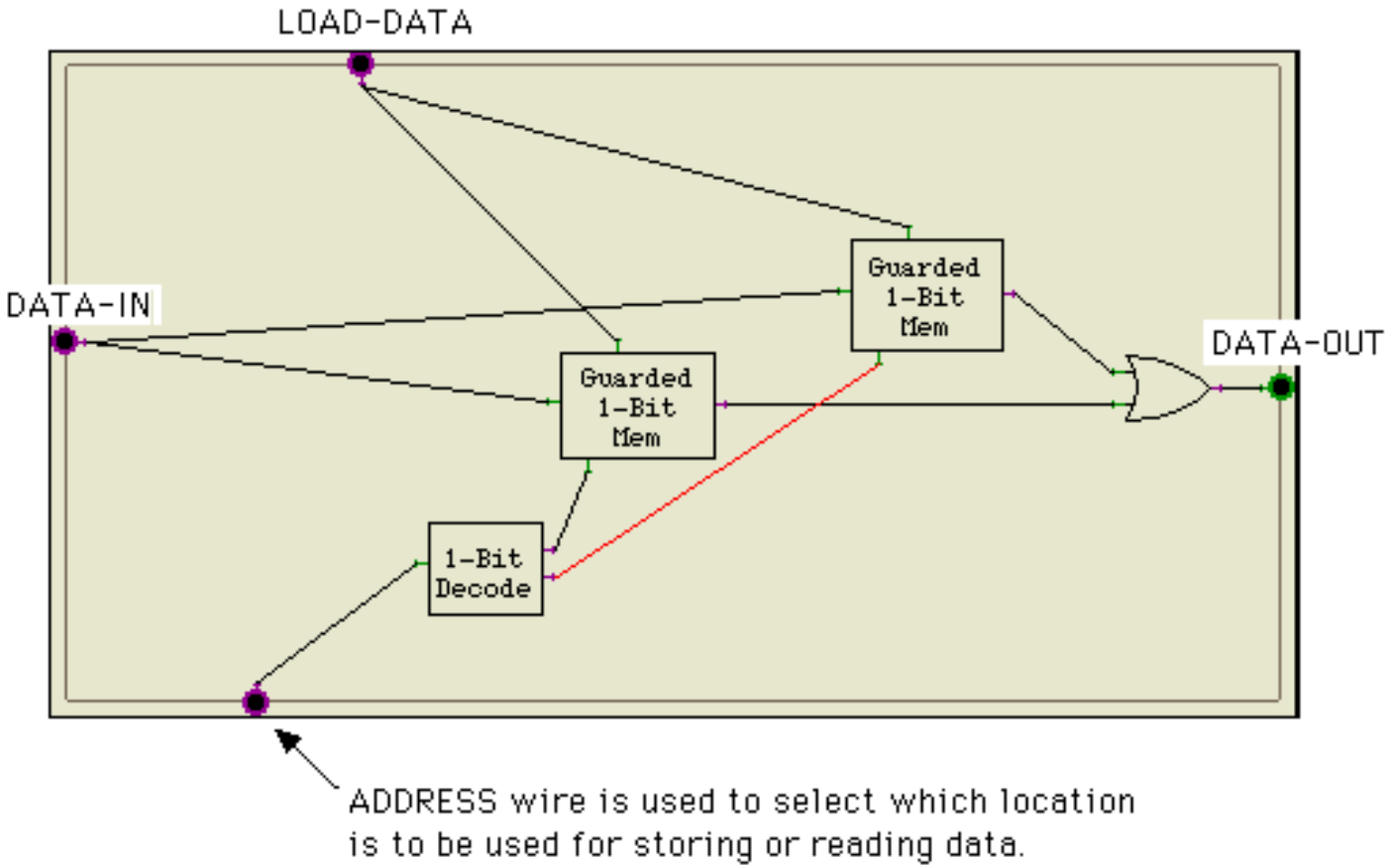
Guarded 1-bitine mälukiip

- Ekstra lüliti kiibi sisse või väljalülimiseks



RAM

- Random-access memory



Ecki xComputer

- http://math.hws.edu/TMCM/java/labs/xComputerLab1.html
- Arvuti põhiosade (protsessor + mälu) simulatsioon väikese Java programmiga.
- Käsusüsteem sarnaneb väga esimeste päris-mikroprotsesoritega
- Lihtsama arusaadavuse tõttu kasutab kahebaidiseid mälupesi (16 bitti), mitte ühebaidiseid, nagu harilik arvuti.
- Mälu on 1024 pesa (1 K), seega 2 Kbaiti.
- Aadressi jaoks kasutusel 10 bitti.
- Esimestel koduarvutitel oli ka 4-16 Kbaiti (umbes sama hulk mälu)

Ecki xComputer ...

- Olulist: protsessori sees on väike hulk spetsiaal-mälupesi (registrid)
- Tehteid saab teha ainult nende registrite vahel.
- Ei ole näiteks võimalik liita otse kahte mälus olevat arvu: enne tuleb nad registritesse kopeerida, siis seal liita, siis tulemusregistrist (nn akumulaator) mäll kirjutada.
- Koha, kust mälust loetakse/kirjutatakse näitab ADDR register.
- Koha, kust lugeda järgmine käsk, näitab PC (program counter) register

Käskude täitmine

- **Kaks tsüklit üksteise sees:**

- **Välimine tsükkel** suurendab igal ringil PC-d (program counterit), st igal ringil võetakse täidetav käsk järgmisest mälupesast.

  - **Sisemine tsükkel** toimub iga käsu sees. Sisemise tsükli jooksul täidetakse käsu sisemisi pisi-samme.

    Üks pisi-samm vastab mingile juhtmele voolu peale andmisele, mispeale käivitub vastav loogika-ahel protessoris ja selle tulemus salvestatakse mõnda registrisse.

- **Masina taktsagedus** on see sagedus, kui tihti pisi-samme täidetakse. Iga järgmise pisi-sammu alustamise jaoks on masinas kell, mis annab kindla sagedusega impulsse. Pisi-sammu number saadakse nende impulsside kokkulugemisega.

xComputer-i põhiregistrid

- The **X and Y registers** hold two sixteen-bit binary numbers that are used as input by the ALU. For example, when the CPU needs to add two numbers, it must put them into the X and Y registers so that the ALU can be used to add them.

- The **AC** register is the accumulator. It is the CPU's "working memory" for its calculations. When the ALU is used to compute a result, that result is stored in the AC. For example, if the numbers in the X and Y registers are added, then the answer will appear in the AC. Also, data can be moved from main memory into the AC and from the AC into main memory.

- The **FLAG** register stores the "carry-out" bit produced when the ALU adds two binary numbers. Also, when the ALU performs a shift-left or shift-right operation, the extra bit that is shifted off the end of the number is stored in the FLAG register.

... Registrid ...

- The **ADDR** register specifies a location in main memory. The CPU often needs to read values from memory or write values to memory. Only one location in memory is accessible at any given time.

- The **PC** register is the program counter. The PC specifies the location in memory that holds the next instruction to be executed.

- The **IR** is the instruction register. When the CPU fetches a program instruction from main memory, this is where it puts it. The IR holds that instruction while it is being executed.

- The **COUNT** register counts off the steps in a fetch-and-execute cycle. It takes the CPU several steps to fetch and execute an instruction. When COUNT is 1, it does step 1; when COUNT is 2, it does step 2; and so forth. Remember that as the COUNT register counts 0, 1, 2,..., just one machine language program is being executed

Hierarhia pistikutest progekeelteni

- Esimene: programmeerimismeetod: kaablid ja pistikud
- Teine: von Neumanni arhitektuur, programm mälus binaarkoodina:

```
01011101 01001011 01010101 11010101 10101001 ....
```

  Lihtsam kirjutada hexas, ntx `4A FC 09 B2 ....`

- Kolmas: Esmane progekeel: assembler.

  Üks masinakäsk: tüüpiliselt üks rida assembleriprogrammi

- Neljas: Harilik progekeel ehk nn kõrgkeel (fortran, basic, c, java, python jne jne).

  Harilikud valemid, if-then-else jne, a la `x=2*y+sin(y);`

Ecki assembler

- This program counts. It starts by putting the number 1 into memory location 12, and then it adds one to the number in that location over and over, forever.

```
LOD-C 1
STO 12
LOD 12
INC
STO 12
JMP 2
```

```
          LOD-C 1      ; Set Count equal to 1
          STO Count
Loop:     LOD Count    ; Add 1 to Count
          INC
          STO Count
          JMP Loop     ; Jump back to start of loop

          @12
Count:    data ;  Location to be used for counting
```

Sumto MIPS-I (SGI spinoff) assembleris

- Argumendid registritesse $4 ja $8
- Resultaat registrisse $2

```
sumto:                      ; Register $4 on n
        li    $3, 0         ; Register $3 on summa
        li    $2, 0         ; Register $2 on i
        blt   $4, $0, L3    ; Kui n<0 mine L3
L5:     addu  $3, $3, $2    ; sum = sum + i
        addu  $2, $2, 1     ; i = i + 1
        ble   $2, $4, L5    ; Kui i<=n mine L5
L3:     move  $2, $3        ; Sum sisaldab resultaati.
        Jr    $31           ; Mine aadressile registris $31
```

Sumto ja Sun Sparc'i assembler

- Sparc saadab argumendid registrites %o0 kuni %o7 ja resultaadi %o0
- Instruktsioon peale hüpet tehakse alati

```
_sumto:                     ; Register %o0 on n.
        mov %o0,%g3         ; Salvesta n registrisse %g3.
        mov 0,  %o0         ; Register %o0 on nyyd sum.
        cmp %o0,%g3         ; Kui 0>n ...
        bg  L3              ; ... mine L3
        mov 0,  %g2         ; ,aga enne i=0.
        add %o0,%g2,%o0     ; sum = sum + i.
L5:     add %g2,1  ,%g2     ; i = i + 1.
        cmp %g2,%g3         ; Kui i<=n ...
        ble,a L5            ; ... mine L5
        add %o0,%g2,%o0     ; ,aga enne sum = sum + i.
L3:     retl                ; Valmis...
nop                         ; ,aga enne ära tee midagi!
```

Sumto ja Intel 386, 486, Pentium, ...

- 386 on vähe registreid, argument saadetakse hariliku mälu kaudu. Resultaat saadetakse registris %edx.

```
_sumto:
        pushl %ebp              ; Loome ''framepointer''-i
        movl  %esp,%ebp         ;
        movl  8(%ebp),%ecx      ; Võta n.
        xorl  %eax,%eax         ; sum = 0
        xorl  %edx,%edx         ; i = 0
        cmpl  %ecx,%eax         ; Kui i>n ...
        jg L3                   ; ... mine L3
        .align 2
L5:     addl  %edx,%eax         ; sum = sum + i
        incl  %edx              ; i = i+1
        cmpl  %ecx,%edx         ; Kui i<=n ...
        jle L5                  ; ... mine L5
L3:     leave                   ; Taasta ebp.
        ret                     ; Valmis!
```

Alamprogrammide kasutamine

```
        lod-c 13  ; Set up to call the subroutine with sto N1
                  ; N1 = 13, N2 = 56, and ret_addr = back.
        lod-c 56
        sto N2
        lod-c back
        sto ret_addr
        jmp Multiply     ; Call the subroutine.

back:   lod Answer   ; When the subroutine ends, it returns
                     ;    control to this location, and the
                     ;    product of N1 and N2 is in Answer.
                     ;    This LOD instruction puts the answer
                     ;    in the accumulator.

        hlt    ; Terminate the program by halting the computer
```